1-1-1994

Storing Logical Form in a Shared-Packed. Forest

Mary P. Harper
Purdue University School of Electrical Engineering

Storing Logical Form in a Shared-Packed Forest

Mary P. Harper

TR-EE 94-6
JANUARY **1994**

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907-1285

Storing Logical Form in a Shared-Packed. Forest

Mary P. Harper
School of Electrical Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285
harper@ecn.purdue.edu

January 31, 1994

Abstract

To compactly represent sentences containing syntactic ambiguity until the necessary information has been processed to refine the meaning, we have modified an all-path context-free grammar parser to generate a shared-packed parse forest annotated with logical form. An annotated shared-packed forest cannot contain every representation of a highly ambiguous sentence without using an intractable amount of space. Hence, our program stores procedure calls for creating all possible logical forms for a constituent in the forest. The resulting forest contains the same number of nodes and is not much bigger than the original forest. Furthermore, the stored procedural information can be used by a program to construct representations for any of the constituents in the forest for subsequent testing against a world model. After performing each test, the program can incrementally prune the forest of ambiguity.

Key Words: natural language processing, parse forest, logical form

1 Introduction

A goal of natural language research is to provide a computer model for understanding English sentences. One approach to building this model is to require the generation of an unambiguous internal representation for each sentence before attempting to represent subsequent sentences. The problem with this approach is that, in order to make the inferences necessary to resolve the ambiguities, some internal representation is needed for both the current sentence as well as subsequent sentences. A more powerful approach is to build a representation for a sentence without resolving the ambiguity and wait until the necessary information has been processed. [10, 11, 1] to update the meaning for that sentence.

There are several types of ambiguity in natural languages including lexical ambiguity, syntactic (or structural) ambiguity, quantifier scope ambiguity, and anaphora or ambiguity of reference. Lexical ambiguity results from the fact that most words have several different (though often related) uses. For example, the word *pen* can have at least two different meanings: it can be a writing instrument or an enclosure for animals or small children. Syntactic ambiguity occurs when a sentence has more than one possible structural analysis. *Fred saw the bird with the binoculars* is an example with two structural analyses, each of which maps to a different meaning: either the bird has the binoculars, or Fred is using the binoculars to see the bird. Quantifier scope ambiguity arises when sentences contain several noun phrases with determiners corresponding to different quantifiers. For example, the sentence *Every dog loves a human* has two different meanings: either every dog loves the same person or each dog loves a potentially different person. Finally, anaphora [12] occurs when sentences contain expressions like pronouns, definite noun phrases, or ellipses, which either have linguistic antecedents or depend on salient individuals in the environment of the speaker or hearer. For example, the sentence *he did* is ambiguous because it contains a pronoun and verb phrase ellipsis.

Ambiguities of the types described above are very common in natural language, and each type must 'be resolved for a natural language understanding program to be effective. Since syntax limits the possible meanings of a sentence (and the words in the sentence), natural language processing programs often analyze the structure of a sentence before attempting to determine its meaning. For example, lexical ambiguity can often be resolved as the sentence is parsed (e.g., the word *ran* in I *ran the computer program* cannot take on the meanings associated with the intransitive form of that verb), but if it is not, additional

information may be required (e.g., to determine the intended meaning of *pen* in I *spilled some paint on the pen* requires more information). Syntactic ambiguity by definition cannot be resolved by parsing; some additional knowledge is needed to select the intended meaning. Ambiguity of reference and quantifier scope ambiguity can often be reduced by using simple type restrictions and syntactic constraints, but again, additional information is required in many cases. Therefore, in general, to refine the meaning of an ambiguous sentence, additional knowledge sources must be used, including selectional restrictions, world knowledge, and contextual information.

The use of world knowledge and contextual information often requires inference, and hence, access to the representations of the sentence and possibly its components. But at the same time, because of ambiguity, a program might not be able to enumerate all of the possible representations for a sentence and its components since just listing all possible structural analyses for syntactically ambiguous sentences can be intractable, and each structural analysis of a sentence produces at least one additional meaning. In this paper, we will focus on the prolblem of efficiently maintaining syntactic ambiguity while determining the logical representation for a sentence. In particular, we describe an approach which combines shared-packed parse forests with semantic construction routines. This approach allows a program attempting to eliminate ambiguity from a sentence to apply higher level knowledge sources to the logical representations of desired constituents in the parse forest (e.g., it could eliminate alternative parses for a noun phrase whose representation does not match objects in a world model).

2 Shared-Packed Packed Parse Forests: A Compact Representation for Syntactically Ambiguous Sentences

Tree structures, called parse trees, are used to represent the structural properties of the sentence. Because language is often syntactically ambiguous, it is common for a particular sentence to have more than one parse tree. For example:

Every man saw the boy with his binoculars.

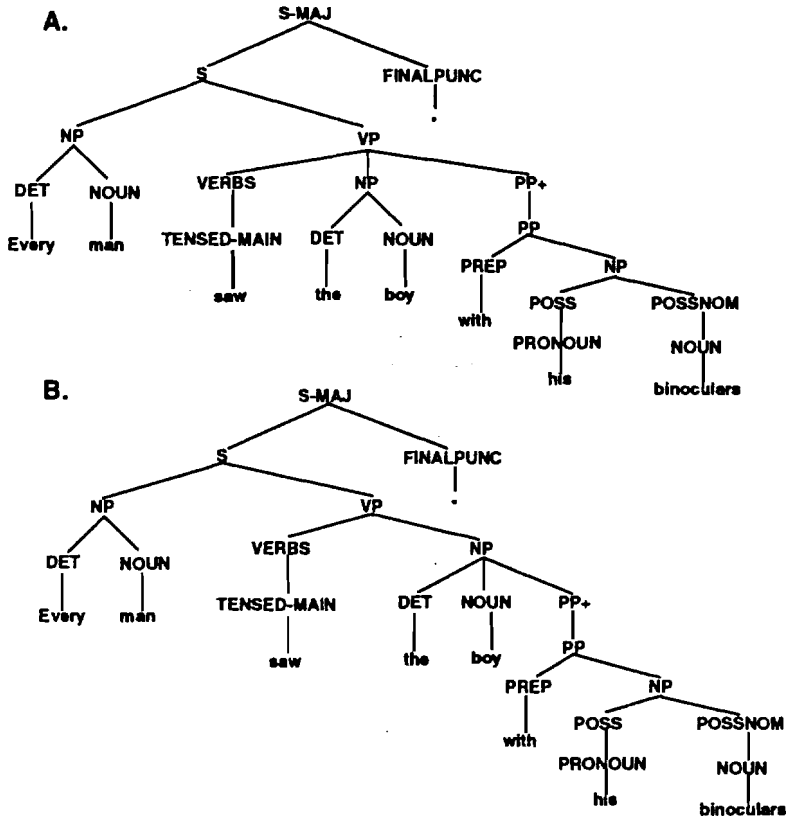

Figure 1: Parse Trees for *Every man saw the boy with his binoculars.*

This sentence has two potential parses, as shown in Figure 1. In the first parse tree (Figure 1A) the prepositional phrase *with his binoculars* is attached to the verb phrase. However, in the other (Figure 1B), it is attached to the object noun phrase. These two structures give rise to very different meanings for the sentence. In the first case, every man is using the binoculars to see the boy; whereas, in the second, the boy has the binoculars.

One way to enable a natural language program to process the meanings of syntactically ambiguous sentences is to incorporate semantic construction routines into a parser that produces each of structural analysis for a sentence, one parse tree at a time, and maps each tree to a separate logical representation. The program must then attempt to determine which meaning for the sentence is the intended one. One problem with this approach is that the nurnber of parse trees produced for some ambiguous sentences is quite large. For example, a parser analyzing sentences with multiple prepositional phrases (PP) can produce an intractable number of possible parses for the sentence. Our example has two different

parse trees, each with a distinct representation, a very manageable number. However, as the number of prepositional phrases in a sentence increases, the number of possible parse trees and their corresponding representations grows as the Catalan numbers [8], where $C_n = \binom{2n}{n}\frac{1}{n+1}$. For sentences with one object and one PP following the verb (i.e., n = 2), there are 2 parses; for one object and 2 post-object PPs (i.e., n = 3), there are 5 parses; and for one object plus four post-object PPs, there are 42 parses. Since the number of parses for a sentence with multiple PPs grows so quickly (i.e., faster than exponential [15]), the time to list out all the possible meanings for each tree can be prohibitive. A one-parse-tree-at-a-time approach which uses no mechanism for storing subresults from a parse (e.g., a chart or parse forest) is also inefficient because it is unable to reuse the results of the semantic and contextual tests made on a subtree of a rejected parse during the evaluation of an alternative parse tree. The need for efficiency dictates the need for another approach to manage the ambiguity of a sentence.

The efficiency of the first approach can be improved by requiring that each indeterminacy in the parse be resolved as soon as it arises [5] to prevent backtracking. To do so, however, requires that enough information be available at that point in the parse to select among the alternatives. This assumption is unjustified in many cases; words occurring later in the sentence or possibly subsequent sentences may be needed to resolve the ambiguity. This approach has been used in a translation system which uses Alshawi's quasi logical form [3]. A slightly different alternative is to work with the highest preference choice only [6, 2]. Though this approach is efficient, it provides the most likely parse for a sentence (usually independently of context), not necessarily the correct parse.

An alternative scheme'for coping with syntactic ambiguity is to change the grammar rules so that they provide a single parse tree for a syntactically ambiguous sentence and then wait for the semantic routines to determine what is possible. The traditional way to write a rule for an NP with noun modifiers is as follows:

```
NP → DET N1
N1 → NOUN
N1 → N1 N1
```

This grammar generates a very large number of possible structures for noun phrases like ***the pretty little girl school;*** however, it also eliminates from consideration impossible noun modifier structures by not allowing crossover between modifiers. For example, the grammar

would never allow a structure such that ***pretty*** modifies ***girl*** which modifies ***.school,*** and ***little*** modifies ***school.*** On the other hand, an alternative rule can be used to generate a single structural analysis for the sentence, as shown below:

NP → DET NOUN* NOUN

This rule ignores the structure of noun modifiers of a head noun, placing them all at the same level in the parse tree. However, without a structure to limit the possible modifier relationships, a semantic routine might incorrectly allow a noun modifier to modify any of the nouns that follow it. To work correctly, the semantic routines would have to encode information already contained in the first set of rules in order to prevent impossible modifications frorn being tested and/or accepted. Another possibility is to use a least commitment grammar which provides only one of the possible modifier structures for an NP:

NP → DET **N1** NOUN
N1 → **NOUN**
N1 → **N1 N1**

To allow an interpretation based on one of the other possible syntactic structures, the semantic routines operating on the output of a least commitment parser must be able to adapt the tree for other interpretations. Pollack and Pereira [17] use a parser that produces a single least-commitment parse tree in a system for handling semantic and pragmatic interpretation.

A fourth approach is to combine an all-path parsing algorithm [9, 14, 7, 21, 19] with routines for generating logical representations in order to create a shared-packed parse forest annotated with the logical representations for the constituents in the forest (i.e., an annotated shared-pached parse forest). Before discussing the benefits of this approach, we will first describe the properties of a shared-packed parse forest [19, 20, 21].

A shared-packed parse forest is a data structure which stores all parses of a sentence in a compact form. Consider the packed parse forest produced by an implementation of Tomita's parser [21] for the sentence ***Every man saw the boy with his binoculars*** shown in Figure 2, along with a picture of the forest shown in Figure **3.** The forest stores both terminal and non-terminal nodes. Non-terminal nodes contain lists of node numbers of the children that make up a parse of that constituent. The start symbol for the grammar is S-MAJ, which in the above example consists of a non-terminal node for an S and a final punctuation terminal

```
24 ((S-MAJ16 S-MAJ) (DOWN (21 23)))
23 ((.8 FINALPUNC .) (DOWN T))
22 ((NP14 NP) (DOWN (7 8 19)))
21 ((S13 S) (DOWN (3 20)))
20 ((VP12 VP) (DOWN (6 9 19) (6 22)))
19 ((PP+11 PP+ NIL) (DOWN (18)))
18 ((PP10 PP NIL) (DOWN (12 17)))
17 ((NP9 NP) (DOWN (16 15)))
16 ((POSS8 POSS) (DOWN (13)))
15 ({POSS-NOM7 POSS-NOM) (DOWN (14)))
14 ((BINOCULARS7 NOUN BINOCULARS) (DOWN T))
13 ((HIS6 PRONOUN HIS) (DOWN T))
12 ((WITH5 PREP WITH) (DOWN T))
11 ((S6 S) (DOWN (3 10)))
10 ((VP5 VP) (DOWN (6 9)))
9  ((NP4 NP) (DOWN (7 8)))
8  ((BOY4 NOUN  BOY) (DOWN T))
7  ((THE3 DET THE) (DOWN T))
6  ((VERBS3 VERBS) (DOWN (5)))
5  ((TENSED-MAIN2 TENSED-MAIN) (DOWN (4)))
4  ((SAW2 VERB SEE) (DOWN T))
3  ((NP1 NP) (DOWN (1 2)))
2  ((MAN1 NOUN MAN) (DOWN T))
1  ((EVERY0 DET EVERY) (DOWN T))
```

Figure 2: The shared-packed parse forest for *Every man saw the boy with his binoculars.*

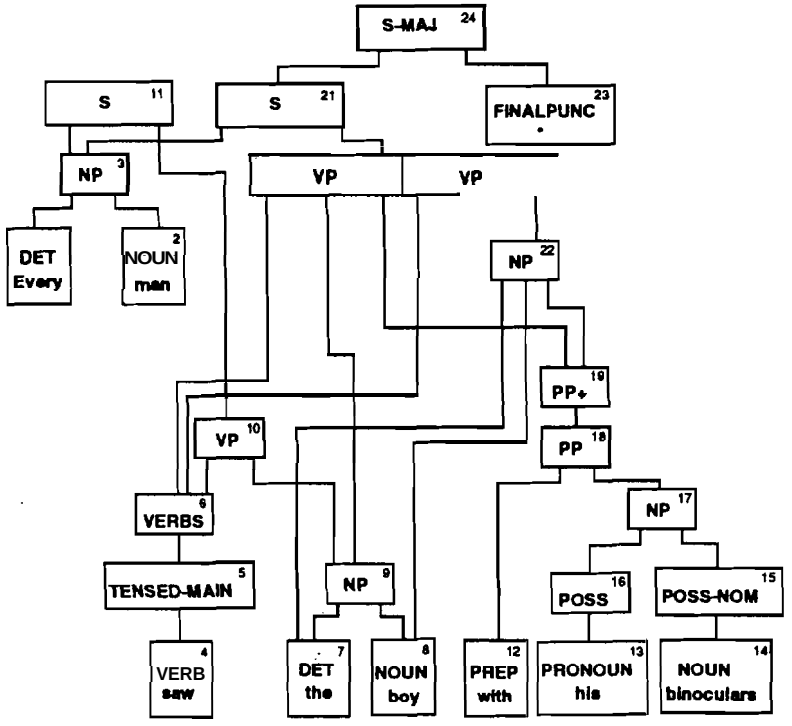

Figure 3: A picture of the shared-packed parse forest from Figure 2.

node. Because a non-terminal node may have descendents with multiple parses, there can be more than one parse tree for the constituent. This results from the fact that the parser packs forest nodes together when they share a common state vertex and have the same features. These packed nodes contain alternative parses for a non-terminal constituent of the parse tree. For example, in Figure 2, the VP with the index of 20 has two parses, one consisting of nodes with indices 6, 9, and 19 and the other with indices 6 and 22. Early in the parse, node 20 had only one set of children, as shown below:

```
20 ((VP12 VP) (DOWN (6 9 19)))
```

Later, after node 22 was created, the parser added (6.22) to the list of children for the VP node. This node packing occurs when the parser is preparing to reduce the phrase consisting of the subtrees 6 and 22 with a VP rule. Since it has the same state vertex on its left and right in the parse stack as the item already stored in the forest, the alternative parse is added to the list of possible children for the already stored constituent. Nodes can appear in the forest that never participate in a sentence parse (e.g., nodes 10 and 11). These useless nodes can easily be pruned after parsing is complete by marking all nodes that participate in a parse beginning with the start symbol, S-MAJ, and freeing those that are unmarked.

Seo and Simmons [19] have introduced syntactic graphs, which are constructed from shared-packed parse forests, to represent ambiguous parses for a sentence. The syntactic graph encodes the modifier links between a head word and its modifiers. An advantage of this approach is that 'words which participate in multiple parses, by modifying different words in different ways, have multiple arcs entering the node. For example, if a preposition (as head of a PP) can modify either a noun or a verb, there would be two arcs entering the node for the preposition, one from the noun and one from the verb. Hence, the point of ambiguity can be pinpointed to the attachment decision. They claim that a parse forest does not give the!same direct access to internal ambiguity; the ambiguous points in a parse forest can be detected only by traversing the forest. Certainly, by examining the forest shown in Figure 2, one cannot immediately detect that the ambiguity for the sentence resides with the PP attachment. However, by adding links between each of the nodes in the forest and its parent node and then pruning the nodes that do not participate in a legal parse for the sentence, the forest does give a better view of this ambiguity (see Figure 4). Nodes with

```
22 ((S-MAJ16 S-MAJ) (DOWN (19 21)) (UP T))
21 ((.8 FINALPUNC .) (DOWN T) (UP 22))
20 ((NP14 NP) (DOWN (7 8 17)) (UP 18))
19 ((S13 S) (DOWN (3 18)) (UP 22))
18 ((VP12 VP) (DOWN (6 9 17) (6 20)) (UP 19))
17 ((PP+11 PP+ NIL) (DOWN (16)) (UP 18 20))   ; attatch to an NP or VP
16 ((PP10 PP NIL) (DOWN (10 15)) (UP 17))
15 ((NP9 NP) (DOWN (14 13)) (UP 16))
14 ((POSS8 POSS) (DOWN (11)) (UP 15))
13 ((POSS-NOM7 POSS-NOM) (DOWN (12)) (UP 15))
12 ((BINOCULARS7 NOUN BINOCULARS) (DOWN T) (UP 13))
11 ((HIS6 PRONOUN HIS) (DOWN T) (UP 14))
10 ((WITH5 PREP WITH) (DOWN T) (UP 16))
9  ((NP4 NP) (DOWN (7 8)) (UP 18))
8  ((BOY4 NOUN BOY) (DOWN T) (UP 9 20))   ; in two different NPs
7  ((THE3 DET THE) (DOWN T) (UP 9 20))    ; in two different NPs
6  ((VERBS3 VERBS) (DOWN (5)) (UP 18))
5  ((TENSED-MAIN2 TENSED-MAIN) (DOWN (4)) (UP 6))
4  ((SAW2 VERB SEE) (DOWN T) (UP 5))
3  ((NP1 NP) (DOWN '(1 2)) (UP 19))
2  ((MAN1 NOUN MAN) (DOWN T) (UP 3))
1  ((EVERY0 DET EVERY) (DOWN T) (UP 3))
```

Figure **4:** The pruned shared-packed parse forest for Every man saw the boy with his binoculars with pointers to parent nodes.

more than one parent participate in multiple parses for a sentence. In the example forest of Figure 4, there are two different NPs that contain the and boy, and the PP+ constituent in node 17 is a member of either an NP or a VP. Though this is not quite as compact as a syntactic graph for the same sentence, it does provide some very useful information on the sources of ambiguity in the sentence. For example, if a noun phrase containing the word boy in Figure 4 can either have a PP+ attached to it (as in node 20) or not (as in node 9), and the world model does not support the attachment, then the forest can be easily pruned of that possibility by deleting all references to the noun phrase at node 20. The deletion process removes node 20 from the up pointers of node 20's children (i.e., nodes 7, 8, and 17) and deletes parses containing node 20 from node 20's parent node, 18. Once the deletion is complete, the forest is no longer ambiguous, as shown in Figure 5.

Hence, rather than transforming the parse forest to a parse graph to represent the syntactic structure for ambiguous sentences, we prefer to store pointers to parent nodes and utilize the shared-packed parse forest to store logical representations. Use of an annotated shared-packed parse forest has the following benefits:

```
22 ((S-MAJ16 S-MAJ) (DOWN (19 21)) (UP T))
21 ((.8 FINALPUNC .) (DOWN T) (UP 22))
20 ((NP14 NP) (DOWN (7 8 17)) (UP 18))          ; can delete node
19 ((S13 S) (DOWN (3 18)) (UP 22))
18 ((VP12 VP) (DOWN (6 9 17)) (UP 19))
17 ((PP+11 PP+ NIL) (DOWN (16)) (UP 18))    ; attatch to a VP
16 ((PP10 PP NIL) (DOWN (10 15)) (UP 17))
15 ((NP9 NP) (DOWN (14 13)) (UP 16))
14 ((POSS8 POSS) (DOWN (11)) (UP 15))
13 ((POSS-NOM7 POSS-NOM) (DOWN (12)) (UP 15))
12 ((BINOCULARS7 NOUN BINOCULARS) (DOWN T) (UP 13))
11 ((HIS6 PRONOUN HIS) (DOWN T) (UP 14))
10 ((WITH5 PREP WITH) (DOWN T) (UP 16))
9  ((NP4 NP) (DOWN (7 8)) (UP 18))
8  ((BOY4 NOUN BOY) (DOWN T) (UP 9))
7  ((THE3 DET THE) (DOWN T) (UP 9))
6  ((VERBS3 VERBS) (DOWN (5)) (UP 18))
5  ((TENSED-MAIN2 TENSED-MAIN) (DOWN (4)) (UP 6))
4  ((SAW2 VERB SEE) (DOWN T) (UP 5))
3  ((NP1 NP) (DOWN (1 2)) (UP 19))
2  ((MAN1 NOUN MAN) (DOWN T) (UP 3))
1  ((EVERY0 DET EVERY) (DOWN T) (UP 3))
```

Figure 5: An unambiguous parse for *Every man saw the boy with his binoculars* given a certain world model.

1. It provides a space savings by' packing duplicated nodes into a single entry in the forest [9, 21, 19], thus reducing the overhead when it is necessary to keep all parses for a sentence around until it is possible to make an informed choice among;the alternative meanings.

2. It provides a direct method for focusing on the points of ambiguity in a sentence when parent links are included for each node.

3. It is able to reuse the semantic decisions made for a subtree of a rejected parse tree. When a node in the forest is limited to a single parse, it is limited for all parses of the sentence containing that node.

Because of these benefits, we have designed a program to generate a shared-packed parse forest annotated with the logical form developed by Harper [10, 11], by augmenting a Tomita-style LR parser with the necessary routines for constructing the logical form representation. Tomita's parser is a bottom-up LR(k)-based parser which constructs a forest of all possible parses of the given input while using a graph-structured stack and breadth-first search to handle non-determinism in the parse of a sentence. In the next section, we describe three

methods for interfacing our logical form routines with Tomita's parser. Despite the fact that we utilized Tomita's parser to construct a specific logical form representation., the conclusions we draw can also be applied to the more efficient parsers [18, 9] that produce other logical representations (e.g., [4, 22, 13]) in even more compact forests [16].

Combining Logical Form with Forests: A Case study

Previously, our logical form routines were interfaced with a-single-parse-tree-at-a-time, top-down ATN parser [10, 11]. This made it relatively easy to create compositional logical form routines and to interface them with the parser. These routines were developed to construct and store tlhe logical forms for each major type of constituent. Some routines created logical representations for the basic constituents like nouns and verbs, while routines for more complex constituents, like verb phrases, noun phrases, and sentences, combined the logical representations of several constituents into a larger representation. Function calls for constructing the logical forms were then added to the arcs in the grammar networks and were executed whenever the arc was successfully traversed (after constituent and feature tests succeeded). Since one parse tree was built at a time, the logical form for each tree was constructed and stored before another tree was produced by the parser's search mechanism; hence, none of the complex logical form routines had to combine more than a single representation far each of its constituents. The logical representations were easy to create in this approach, but unfortunately the parser was intractable because it generated a single parse tree at a time.

In the Tomita parser, the grammar rules consist of production rules containing a left-hand side, a right-hand side, and a set of actions. These actions include feature tests, which must succeed for the reduce operation to proceed, and routines for storing feature values and for constructing and storing the logical form with a node in the forest. For example, the following rule is used to parse a sentence consisting of an NP and a VP:

```
(S -> (NP VP)
      ((== #$NP (get-the person of #$VP))              ; Subject-verb agreement test
       (=! #$PHRASE 'statement)                        ; Set the HOOD of the sentence
       (logical-form 'sentence :np #$NP :vp #$VP)))    ; Create the logical form
```

The left-hand side of this rule is S, and the right-hand side is a list consisting of an NP and a VP. For the rule to succeed during parsing, the right-hand side of the rule must

match, ancl the subject-verb agreement test must return true. If it does, a parse node is created with a list of children consisting of #$NP and #$VP, the node numbers of the two constituents that make up the S. Additionally, the feature information and logical form for the constituent are stored in the node created for the forest[1].

Unlike the ATN parser used by Harper [10, 11], the Tomita parser is a bottom-up, all-path parsing algorithm which creates a parse forest by packing parse nodes together to save space and time. Because nodes with two alternative parses must also produce two different semantic representations, our logical form construction routines must be able to store and retrieve multiple logical forms for ambiguous constituents in the forest. This requirement introduces two problems to solve. First, packed nodes in a forest represent multiple parses, which produce multiple representations; hence, our routines for constructing logical forms for sentences (and other complex constituents) may have to combine multiple representations for each of their constituents. Second, the annotated shared-packed parse forest cannot store every representation of a highly ambiguous sentence without using an intractable amount of space. Any approach which uses a parse forest to store logical representations for the constituents of a sentence will have to address these problems. We will describe three methods for interfacing the LR parser with logical form routines, and illustratc: the differences between these approaches by using the parse of the sentence *Fred saw frogs in cars with Bill.*

The first and simplest method is to prevent two nodes from being packed together (except for the start symbol), if they have different logical forms (see the parse forest below). This approach is similar to the method employed by the ATN to generate the logical forms for a sentence, and is equivalent to mapping individual parse trees to a logical representation.

```
36 ((S-MAJ62 S-MAJ) (DOWN (20 35) (24 35) (26 35) (32 35) (34 35)) (UP T))
   ; Store 5 representations for the S-MAJ by combining the single representations of
   ; each of the node pairs.
35 ((.7 FINALPUNC .) (DOWN T) (UP 36))
34 ((S61 S) (DOWN (2 33)) (UP 36))
   ; Store an S representation by combining the representations in nodes 2 and 33.
33 ((VP60 VP) (DOWN (5 7 29)) (UP 34))
   ; Store a VP representation by combining the representations in nodes 5, 7, and 29.
32 ((S55 S) (DOWN (2 31)) (UP 36))
   ; Store an S representation by combining the representations in nodes 2 and 31.
31 ((VP54 VP) (DOWN (5 30)) (UP 32))
   ; Store a VP representation by combining the representations in nodes 5 and 30.
30 ((NP51 NP) (DOWN (6 29)) (UP 31))
   ; Store 1 representation for an NP by combining the representations in nodes 6 and 29.
```

[1]In examples, we omit the feature information stored on nodes and simply indicate the number of logical forms stored for a node, not the actual representation, to simplify the forest.

29 ((PP+48 PP+) (DOWN (28)) (UP 30 33))
; Store 1 representation for a PP+ given the representation in node 28.
28 ((PP47 PP) (DOWN (8 27)) (UP 29))
; Store 1 representation for the PP by combining the representations in nodes 8 and 27.
27 ((NP46 NP) (DOWN (9 18)) (UP 28))
; Store 1 representation for an NP by combining the representations in nodes 9 and 18.
26 ((S45 S) (DOWN (2 25)) (UP 36))
; Store an S representation by combining the representations in nodes 2 and 25.
25 ((VP44 VP) (DOWN (5 7 21)) (UP 26))
; Store a VP representation by combining the representations in nodes 5, 7, ancl 21.
24 ((S39 S) (DOWN (2 23)) (UP 36))
; Store an S representation by combining the representations in nodes 2 and 23.
23 ((VP38 VP) (DOWN (5 22)) (UP 24))
; Store a VP representation by combining the representations in nodes 5 and 22.
22 ((NP35 NP) (DOWN (6 21)) (UP 23))
; Store 1 representation for an NP by combining the representations in nodes 6 and 21.
21 ((PP+32 PP+) (DOWN (11 18)) (UP 22 25))
; Store 1 representation for a PP+ given the representation in nodes 11 and 18.
20 ((S31 S) (DOWN (2 19)) (UP 36))
; Store an S representation by combining the representations in nodes 2 and 19.
19 ((VP30 VP) (DOWN (5 13 18)) (UP 20))
; Store a VP representation by combining the representations in nodes 5, 13, and 18.
18 ((PP+27 PP+) (DOWN (17)) (UP 19 21 27))
; Store 1 representation for a PP+ given the representation in node 17.
17 ((PP26 PP) (DOWN (14 16)) (UP 18))
; Store 1 representation for the PP by combining the representations in nodes 14 and 16.
16 ((NP25 NP) (DOWN (15)) (UP 17))
; Store 1 representation for an NP given the head noun in node 15.
15 ((FRED6 PROPERNOUN FRED) (DOWN T) (UP 16))
14 ((WITH5 PREP WITH) (DOWN T) (UP 17))
13 ((NP20 NP) (DOWN (6 12)) (UP 19))
; Store 1 representation for an NP given the representations in nodes 6 and 12
12 ((PP+11 PP+) (DOWN (11)) (UP 13))
; Store 1 representation for a PP+ given the representation in node 11.
11 ((PP10 PP) (DOWN (8 10)) (UP 12 21))
; Store 1 representation for the PP by combining the representations in nodes 8 and 10.
10 ((NP9 NP) (DOWN (9)) (UP 11))
; Store 1 representation for an NP given the head noun in node 9.
9 ((CARS4 NOUN CAR) (DOWN T) (UP 10 27))
8 ((IN3 PREP IN) (DOWN T) (UP 11 28))
7 ((NP4 NP) (DOWN (6)) (UP 25 33))
; Store 1 representation for an NP given the head noun in node 6.
6 ((FROGS2 NOUN FROG) (DOWN T) (UP 7 13 22 30))
5 ((VERBS3 VERBS) (DOWN (4)) (UP 19 23 25 31 33))
; Store 1 representation given node 4.
4 ((TENSED-MAIN2 TENSED-MAIN) (DOWN (3)) (UP 5))
; Store 1 representation for the verb in 3.
3 ((SAW1 VERB SEE) (DOWN T) (UP 4))
2 ((NP1 NP) (DOWN (1)) (UP 20 24 26 32 34))
; Store 1 representation for an NP given the head noun in node 1.
1 ((FRED0 PROPERNOUN FRED) (DOWN T) (UP 2))

This method is easy to implement, but it.does not take advantage of the shared-packed parse

forest for compactly storing the logical forms. And because different structural variations typically map to different logical representations, the number of nodes in the forest can be exponential (or worse) for some ambiguities.

The second approach is to store the logical representation directly in the shared-packed parse forest:, as shown below:

```
26 ((S-MAJ196 S-MAJ) (DOWN (20'25)) (UP T))
25 ((.7 FINALPUNC .) (DOWN T) (UP 26))
24 ((PP174 PP) (DOWN (8 23)) (UP 21))
   ; Store 1 representation for the PP by combining the representations in nodes 8 ancl 23.
23 ((NP165 NP) (DOWN (9 18)) (UP 24))
   ; Store I representation for an NP by combining the representations in nodes 9 and 18.
22 ((NP130 NP) (DOWN (6 21)) (UP 19))
   ; Store 2 representations for an NP by combining the representations in nodes 6 and 21.
21 ((PP+113 PP+) (DOWN (11 18) (24)) (UP 22 19))
   ; Store 2 representations of PP+ by combining the representations of nodes 11 and 18,
   ; and using the representation of node 24.
20 ((S106 S) (DOWN (2 19)) (UP 26))
   ; Store ti representations of S by combining the representation of node 2 with the
   ; 5 representations of node 19.
19 ((VP95 VP) (DOWN (5 13 18) (5 22) (5 7 21)) (UP 20))
   ; Store 5 representations for the VP, one by combining the representations
   ; in nodes 5, 13, and 18, two by combining the representation in node 5
   ; with the two representations in node 22, and one by combining the representation
   ; of node 5 with the representation of node 7 and the two representations of node 21.
18 ((PP+84 PP+) (DOWN (17)) (UP 19 21 23))
   ; Store 1 representation for a PP+ given the representation in node 17.
17 ((PP79 PP) (DOWN (14 16)) (UP 18))
   ; Store 1 representation for a PP given the representations in node 14 and 16.
16 ((NP76 NP) (DOWN (15)) (UP 17))
   ; Store I representation for an NP given the head noun in node 15.
15 ((BILL5 PROPERNOUN BILL) (DOWN T) (UP 16))
14 ((WITH5 PREP WITH) (DOWN T) (UP 17))
13 ((NP56 NP) (DOWN (6 12)) (UP 19))
   ; Store I representation for an NP by combining the representations in nodes 6 and 12.
12 ((PP+31 PP+) (DOWN (11)) (UP 13))
   ; Store 1 representation for a PP+ given the representation in node 11.
11 ((PP26 PP) (DOWN (8 10)) (UP 12 21))
   ; Store 1 representation for the PP by combining the representations in nodes 8 and 10.
10 ((NP23 NP) (DOWN (9)) (UP 11))
   ; Store 1 representation for an NP given the head noun in node 9.
9  ((CARS4 NOUN CAR) (DOWN T) (UP 10 23))
8  ((IN3 PREP IN) (DOWN T) (UP 11 24))
7  ((NP10 NP) (DOWN (6)) (UP 19))
   ; Store 1 representation for an NP given the head noun in node 6.
6  ((FROGS2 NOUN) (DOWN T) (UP 7 13 22))
5  ((VERBS7 VERBS) (DOWN (4)) (UP 19))
   ; Store I representation given node 4.
4  ((TENSED-MAIN4 TENSED-MAIN) (DOWN (3)) (UP 5))
   ; Store I representation for the verb in 3.
```

```
3 ((SAW1 VERB SEE) (DOWN T) (UP 4))
2 ((NP1 NP) (DOWN (1)) (UP 20))
  ; Store 1 representation for an NP given the head noun in node 1.
1 ((FRED0 PROPERNOUN) (DOWN T) (UP 2))
```

The syntactic ambiguity in the structure of a child node must be reflected in the logical representation of the ancestor nodes. If a parent node consists of two constituents, one with three logical forms and another with two, the construction routines must be able to store the six logical forms for that constituent. This requires that the logical form routines be constructed to combine the logical forms for constituents with more than a single representation. Node packing provides another challenge. When a new node is packed with an already existing node in the forest, the logical representation for the new structure must be stored for that constituent. Also, all of the ancestors of a newly packed node must update their lists of logical representations to reflect the addition of the new parse since packing of a node can occur after many of its ancestors are already members of the forest. In our example, when the second parse is added to node 21, a second logical form would have to be added to logical form list for that node and to the logical form lists of each of its ancestors that are already stored (i.e., 19, 20, and 22) for the forest to be complete.

In contrast to the first approach, this method does not increase the number of nodes in the parse forest; however, an exponential number of logical representations can be created for sentences in some ambiguous grammars. Some space savings can be achieved by using pointers to the representations of a child node when creating the representations for a parent node, because many of the nodes (and corresponding logical forms) in a parse forest are shared by multiple parses. However, for multiple logical representations to share the logical representation of a child node, that representation cannot be affected by the process of constructing the logical form for the parent node. This assumption does not always hold; in some logical representations (e.g., [10, 11]), a constituent's representation is affected by the parse tree containing it. In such a case, the logical form of a shared node would require copying and modification before being used in the logical representation of a parent node. If all logical representations are copied and modified as they are combined into higher logical representations, the parse forest could grow quite large. However, even if the elements of a logical representation do not require copying, the number of representations created for a sentence using the second approach is precisely the number of parses for the sentence, which can be exponential in number.

A third approach is to store a procedure call for creating the logical form of a constituent in the forest and to put off the creation of the logical form, as shown below:

```
26 ((S-MAJ196 S-MAJ) (DOWN (20 25)) (UP T) (CREATGSMAJ-LF :S 20 :PUNC 25))
25 ((.7 FINALPUNC .) (DOWN T) (UP 26))
24 ((PP174 PP) (DOWN (8 23)) (UP 21) (CREATGPP-LF :PREP 8 :OBJ 23))
23 ((NP165 NP) (DOWN (9 18)) (UP 24) (CREATE-NP-LF :NOUN 9 :POSTNOUN-MODS 18))
22 ((NP130 NP) (DOWN (6 21)) (UP 19) (CREATGNP-LF :NOUN 6 :POSTNOUN-MODS 21))
21 ((PP+113 PP+) (DOWN (11 18) (24)) (UP 22 19)
   (CREATE-PP+-LF :PP 11 :PP+ 18) (CREATE-PP+-LF :PP 24))
20 ((S106 S) (DOWN (2 19)) (UP 26) (CREATES-LF :NP 2 :VP 19))
19 ((VP95 VP) (DOWN (5 13 18) (5 22) (5 7 21)) (UP 20)
   (CREA'TGVP-LF :VERB 5 :OBJ1 13 :PP+ 18 :SUBCAT 'TRANS)
   (CREA'TGVP-LF :VERB 5 :OBJ1 22 :SUBCAT 'TRANS)
   (CREA'TGVP-LF :VERB 5 :OBJ1 7 :PP+ 21 :SUBCAT 'TRANS))
18 ((PP+84 PP+) (DOWN (17)) (UP 19 21 23) (CREATE-PP+-LF :PP 17))
17 ((PP79 PP) (DOWN (14 16)) (UP 18) (CREATE-PP-LF :PREP 14 :OBJ 16))
16 ((NP76 NP) (DOWN (15)) (UP 17) (CREATGPROPERNOUN-LF :PROPERNOUN 15))
15 ((BILLS PROPERNOUN BILL) (DOWN T) (UP 16))
14 ((WITH5 PREP WITH) (DOWN T) (UP 17))
13 ((NP56 NP) (DOWN (6 12)) (UP 19) (CREATGNP-LF :NOUN 6 :POSTNOUN-MODS 12))
12 ((PP+31 PP+) (DOWN (11)) (UP 13) (CREATE-PP+-LF :PP 11))
11 ((PP26 PP) (DOWN (8 10)) (UP 12 21) (CREATGPP-LF :PREP 8 :OBJ 10))
10 ((NP23 NP) (DOWN (9)) (UP 11) (CREATGNP-LF :NOUN 9))
9  ((CARS4 NOUN CAR) (DOWN T) (UP 10 23))
8  ((IN3 PREP IN) (DOWN T) (UP 11 24))
7  ((NP10 NP) (DOWN (6)) (UP 19) (CREATGNP-LF :NOUN 6))
6  ((FROGS2 NOUN) (DOWN T) (UP 7 13 22))
5  ((VERBS7 VERBS) (DOWN (4)) (UP 19) (CREATGVERBS-LF :VERBS 4))
4  ((TENSED-MAIN4 TENSED-MAIN) (DOWN (3)) (UP 5) (CREATGVERB-ONLY-LF :VERB 3))
3  ((SAW1 VERB SEE) (DOWN T) (UP 4))
2  ((NP1 NP) (DOWN (1)) (UP 20) (CREATE–PROPERNOUN-LF :PROPERNOUN 1))
1  ((FRED0 PROPERNOUN) (DOWN T) (UP 2))
```

Once the parse forest is complete, the logical form for any node can be created by evaluating the stored procedure call. If the node has multiple parses, then multiple logical forms will be created by the routines automatically. Hence, the logical form routines used in this approach must be able to properly combine the multiple logical forms for its constituents (just as in the second approach). All of the logical forms for the sentence can be producecl by evaluating the logical form routine stored with the S-MAJ, resulting in a potentially large number of representations. However, since, the procedure calls stored with any of the nodes in the forest can be evaluated as needed, the program is able to generate and examine only the representations of the constituents associated with points of ambiguity. This feature can be used by a semantic processing module to determine which of the NPs in the forest make sense given the world model. For example, the head nouns in nodes 6 and 9 have pointers

Number of PPs	Number of Parses	Forest Size for No LF	Forest Size. for Method 1	Forest Size for Method 2	Forest Size for Method 3
0	1	1,184 (11)	3,450 (11)	3,450 (11)	1,353 (11)
1	2	1,803 (19)	9,971 (21)	9,748 (19)	2,200 (19)
2	5	2,660 (30)	32,043 (42)	30,704 (30)	3,370 (30)
3	14	3,703 (44)	108,291 (93)	107,716 (44)	5,043 (44)
4	42	5,003 (61)	389,499 (233)	388,671 (61)	7,114 (61)
5	132	6,526 (81)	1,477,644 (651)	1,475,418 (81)	9,658 (81)

Figure 6: A comparison of the size of the parse forest in bytes and number of nodes given the method of constructing the logical representation.

to 5 different NPs, whose logical forms can be created and tested against the model.

The previously described methods for generating logical form were each implemented and evaluated. Figure 6 compares the memory size of the forests in bytes along with the number of nodes generated in the forest (shown in parentheses) for each of the three methods. As a baseline, we also include the number of nodes and the size of the forest when no logical representation is constructed. The third method is superior to the other two methods because: the forest contains the same number of nodes as the original forest without logical form; the size of the forest augmented with logical form is much closer to the size of the original forest; and the logical forms for any of the nodes in the forest can still be accessed by executing the function call(s) stored in that node, providing a flexible tool for higher level processing.

4 Conclusion

To compactly represent sentences containing syntactic ambiguity until the necessary information has been processed to refine the meaning, we have modified an all-path context-free grammar parser to generate a shared-packed parse forest annotated with function calls to create the logical representations for the constituents of the forest. The forest augmented with procedure calls to construct logical representations is a compact data structure containing multiple sentence parses and access to their corresponding logical representations. Hence, a program using this data structure can store the representation in memory for more extensive processing. Once constructed, this annotated shared-packed forest can be utilized by a higher-level module to provide logical representations for pieces of the sentence or for

the entire sentence. Points of ambiguity are easy to detect in the annotated forest because of the presence of multiple parent links. In case of ambiguity, representations for the ambiguous constituents of the sentence can be constructed and tested for validity against a world model, and the annotated forest can then be pruned incrementally.

References

[1] J. Allen. Natural language, knowledge representation, and logical form. Technical Report 367, University of Rochester, Computer Science, Rochester, NY, 1991.

[2] H. Alshawi. Resolving quasi logical forms. Computational Linguistics, 16:133–144, 1990.

[3] H. Alshawi, D. Carter, M. Rayner, and B. Gamback. Translation by quasi logical form transfer. In The Proceedings of the 29th Annual Meeting of the Association for *Computational* Linguistics, 1991.

[4] H. Alshawi and R. Crouch. Monotonic semantic interpretation. In The Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, 1992.

[5] E. Briscoe. Modelling Human Speech Comprehension: A Computational Approach. Ellis Horwood and Wiley, 1987.

[6] T. Briscoe and J. Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. Computational Linguistics: 19:25–59, 1993.

[7] D. Chester. A parsing algorithm that extends phrases. American Journal of Computational Linguistics, 6:87–96, 1980.

[8] K. Church and R. Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. Computational Linguistics, 8:139–149, 1982.

[9] J. Early. An efficient context-free parsing algorithm. Communications of the ACM, 13:94–102, 1970.

[10] M.P. Harper. The representation of noun phrases in logical form. PhD thesis, Brown University, 1990.

[11] M.P. Harper. Ambiguous noun phrases in logical form. Computational Linguistics, 18:419–465, 1992.

[12] G. Hirst. Anaphora in Natural Language Understanding: A Survey. Springer-Verlag, New York, 1981.

[13] G. Hirst. Semantic Interpretation and the Resolution of Ambiguity. Cambridge University Press, Cambridge, England, 1987.

[14] M. Kay. Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, Xerox Corporation, Palo Alto, CA, 1980.

[15] D. E. Knuth. The Art of Computer Programming, volume I. Addison-Wesley, Reading, MA, 1975.

[16] M.J. Nederhof. Generalized left-corner parsing. In Sixth Conference of the European *Chapter* of the Association for Computational Linguistics, Proceedings of the Conference, pages 305–314, Utrecht, The Netherlands, April 1993.

[17] M. E. F'ollack and F.C.N Pereira. An integrated framework for semantic and pragmatic interpretation. In The Proceedings of the 26th Annual Meeting of the .Association for Computational Linguistics, pages 75–86, 1988.

[18] Y. Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In The Proceedings of the 29th Annual Meeting of the Association for *Computational* Linguistics, 1991.

[19] J. Seo and R. F. Simmons. Syntactic graphs: A representation for t'he union of all ambiguous parse trees. Computational Linguistics, 15:19–32, 1989.

[20] M. Tomita. An efficient context-free parsing algorithm for natural languages. In Proceedings of the 9th International Joint *Conference on* Artificial Intelligence, 1985.

[21] M. Tornita. *Efficient* Parsing for Natural Language. Kluwer Academic Publishers, Boston, MA, 1985.

[22] R. Weischedel. A hybrid approach to representation in the JANUS natural language processor. In The Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, pages 193–202, 1989.